RedTyp: A Database of Reduplication with Computational Models

Hossep Dolatian and **Jeffrey Heinz**
Linguistics Department &
Institute for Advanced Computational Science
Stony Brook University
Stony Brook, NY 11790, USA
{hossep.dolatian,jeffrey.heinz}@stonybrook.edu

Abstract

Reduplication is a theoretically and typologically well-studied phenomenon, but there is no database of reduplication patterns which include explicit computational models. This paper introduces RedTyp, an SQL database which provides a computational resource that can be used by both theoretical and computational linguists who work on reduplication. It catalogs 138 reduplicative morphemes across 91 languages, which are modeled with 57 distinct finite-state machines. The finite-state machines are 2-way transducers, which provide an explicit, compact, and convenient representation for reduplication patterns, and which arguably capture the linguistic generalizations more directly than the more commonly used 1-way transducers for modeling natural language morphophonology.

1 Introduction

Reduplication is a cross-linguistically well-attested and ubiquitous morphological operation (Rubino, 2005). The World Atlas of Language Structure (WALS) database documents that 313 out of 368 languages (85%) productively use some form of reduplication to mark one or more semantic functions (Rubino, 2013).[1]

The typology of reduplication can be roughly divided into total reduplication and partial reduplication. Total reduplication copies unboundedly many segments which form some morphological constituent (e.g. a word, stem, root, etc.) as shown in (1a). Partial reduplication copies a bounded number of segments. In partial reduplication, the shape of the reduplicant is most commonly CV (2a), CVC (2b), or CVCV (2c).

[1] Here, the term 'reduplication' is used loosely to mean any morphosyntactic process or morpheme which systematically copies some segmental material from the stem (Hurch, 2005 ff.).

1. Indonesian (Cohn, 1989, 185)
   (a) wanita → wanita∼wanita
   'woman' → 'women'
2. Pangasinan (Rubino, 2005, 11)
   (a) too → to∼too
   'man' → 'people'
   (b) baley → bal∼baley
   'town' → 'towns'
   (c) manok → mano∼manok
   'chicken' → 'chickens'

There is a much more diverse typology than these relatively simple patterns. Typologists have documented various patterns of reduplication which are both common and uncommon (Moravcsik, 1978; Rubino, 2005; Inkelas and Zoll, 2005; Hurch, 2005). Interested readers are referred to Raimy (2011), Urbanczyk (2007), and Inkelas and Downing (2015) for overviews.

Reduplication is difficult to model with existing finite-state tools for two reasons. First, copying an unbounded number of segments (total reduplication) cannot be done by 1-way Finite-State Transducers (1-way FSTs) (Roark and Sproat, 2007), which are overwhelmingly used in computational linguistics (Mohri, 1997; Beesley and Karttunen, 2003). Instead, existing finite-state tools approximate total reduplication by essentially treating it as memorizing a list of existing words in the language (Hulden, 2009a; Hulden and Bischoff, 2009; Cohen-Sygal and Wintner, 2006; Roark and Sproat, 2007). If total reduplication were rare, perhaps this difficulty could be overlooked. However, total reduplication is the most common reduplicative process and it occurs in an estimated 75% of the world's languages (Rubino, 2013).

Second, while 1-way finite-state transducers can copy boundedly many segments (partial reduplication), the number of states needed can be quite large (Roark and Sproat, 2007; Hulden, 2009a; Chandlee and Heinz, 2012; Chandlee, 2017). This can can make them difficult to design and debug. Consequently, there are few (if any) computational resources which model reduplication in a way that is simple, small, easy to design, and linguistically motivated.

Against this background, this paper makes two contributions. First, it introduces a SQL database, which we call RedTyp, of 138 reduplicative processes from 91 languages.[2] These were gathered from various typological surveys of reduplication. We mainly used Moravcsik (1978), a classic survey on reduplication, and supplemented it with other published linguistic surveys (Rubino, 2005; Inkelas and Downing, 2015), with case studies gleaned from other smaller surveys that were narrower in scope e.g. McCarthy and Prince (1995), among others.

A copy of RedTyp exists online at our GitHub page: `github.com/jhdeov/RedTyp` and is available to the public under a Creative Commons non-commercial license (CC BY-NC 4.0).

Second, RedTyp models reduplicative processes with an understudied and under-used type of finite-state technology: 2-way deterministic finite-state transducers (2-way FSTs) (Engelfriet and Hoogeboom, 2001; Filiot and Reynier, 2016). As we explain in Subsection 2.2, and in detail in Dolatian and Heinz (2018b), 2-way FSTs can reread parts of the input string, unlike 1-way FSTs. In addition to allowing 2-way FSTs to model total reduplication exactly, this additional capacity significantly reduces the number of states needed to model partial reduplication. Consequently, 2-way FSTs for reduplication are easy to design, debug, and manage. Besides their state efficiency and practical utility, 2-way FSTs likewise capture the intensional description of reduplication. For more discussion of the role of 2-way FSTs as computational models of reduplication, see Dolatian and Heinz (2018b).

For each reduplicative process in RedTyp, we manually wrote a 2-way FST representing it. In total, we modeled 138 reduplicative processes in RedTyp with 57 2-way FSTs. The average number of states in these machines is 8.82.[3] These FSTs are included in RedTyp, along with a Python script for using them.

The remainder of this paper is organized as follows. Although 2-way FSTs have been studied since the 1960s (Aho et al., 1969), they are a relatively unknown finite-state device in computational linguistics; outside of linguistics, there has been recently been more industrial applications for computational models equivalent to 2-way FSTs (Alur and Černý, 2011; Alur et al., 2014). In order to explain the resource RedTyp, which uses 2-way FSTs, we briefly[4] introduce and define 2-way FSTs in Section 2. Section 3 details the SQL structure of RedTyp, the software implementation of the 2-way FSTs, and the accompanying Python script. Section 4 discusses various aspects of RedTyp including a a high-level comparison of 2-way FSTs to other formal devices, a comparison of RedTyp to the only other reduplication database that exists to our knowledge (the Graz Database on Reduplication (Hurch, 2005)), the utility of RedTyp, and future research directions. Conclusions are in Section 5.

2 Reduplication with two-way finite-state transducers

Informally, both 1-way FSTs and 2-way FSTs can be thought of as machines which read their input from an input tape and write their output onto an output tape (Filiot and Reynier, 2016). In the case of 1-way FSTs, the machine only moves across the input tape from left-to-right and likewise writes the output from left-to-right. Like 1-way FSTs, 2-way FSTs also only write the output from left-to-right. However, unlike 1-way FSTs, 2-way FSTs can move back and forth along the input tape. Because of this difference, they are more expressive than 1-way FSTs. In particular, 2-way FSTs can model total reduplication exactly while 1-way FSTs cannot. In this section, we introduce them informally with an example and then provide a formalization.

2.1 Illustrating 2-way FSTs

The back and forth movement along the input tape is controlled by the transitions with a directional

[2]The name *RedTyp* is in homage to the "Typ" databases including the StressTyp databases (Goedemans et al., 1996; Heinz et al., 2016) and others described in Everaert et al. (2009).

[3]The largest 2-way FST in this sample, from verbal reduplication in Kinande (Downing, 2000), has 29 states. This pattern depends on the size of the root and the number and type of suffixes and prefixes around it. In contrast, we estimate a deterministic 1-way FST would require over 1,000 states for this pattern of partial reduplication.

[4]A fuller technical illustration can be found in Dolatian and Heinz (2018b), and a more informal one in Dolatian and Heinz (In press.).

parameter *d* which specifies whether the machine advances left-to-right (a value of +1), stays put (a value of 0), or moves right-to-left (a value of -1). For a 1-way FST, the value of *d* is always +1. We further assume that input strings are flanked with the left and right boundary symbols, $\rtimes$ and $\ltimes$, respectively.

To illustrate how a 2-way FST works, consider the case of total reduplication in Indonesian. In Indonesian, plurality is marked by reduplicating the entire word or input (3). Data are from Cohn (1989, 185).

3.	(a)	buku	→	buku∼buku
		'book'	→	'books'
	(b)	wanita	→	wanita∼wanita
		'woman'	→	'women'
	(c)	maʃarakat	→	maʃarakat∼maʃarakat
		'society'	→	'societies'
	(d)	kəkuraŋan	→	kəkuraŋan∼kəkuraŋan
		'lack'	→	'lacks'

This total reduplicative process cannot be modeled with a 1-way FST (Roark and Sproat, 2007), but can be easily modeled with a deterministic 2-way FST as in Figure 1. In Figure 1, transitions between states are labeled (i, o, d) where i is the input symbol, *o* is the output string, and *d* is the directional parameter. Σ represents any segment in the input which is not the left-edge or right-edge boundary. The empty string is represented by λ. The boundary symbol $\sim$ in the output plays no crucial function; it visualizes the boundary between the two copies.

To illustrate, Table 1 shows the derivation of /buku/→[buku∼buku] using the 2-way FST in Figure 1. Each row in the table consists of four parts: *input string, output string, current state, transition*. In the *input string*, we underline the input symbol which the 2-way FST will read next. The *output string* is what the 2-way FST has outputted up to that point. The symbol λ marks the empty string. The *current state* is what state the 2-way FST is currently in. The *transition* represents the used transition arc from input to output along with a direction value. In the first tuple, there is no transition arc used (N/A). But for other tuples, the form of the arc is:

$$\textit{input state} \xrightarrow[\text{direction}]{\text{input symbol:output string}} \textit{output state}$$

2.2 Defining 2-way FSTs

Here, we give a formal definition of deterministic 2-way FSTs, synthesizing definitions from Filiot and Reynier (2016) and Shallit (2008). Inputs to a 2-way FST are flanked with the start ($\rtimes$) and end boundaries ($\ltimes$). This larger alphabet is denoted by $\Sigma_{\ltimes}$.

4. **Definition**: A 2-way, deterministic FST is a six-tuple $(Q, \Sigma_{\ltimes}, \Gamma, q_0, F, \delta)$ such that:
   - Q is a finite set of states,
   - $\Sigma_{\ltimes} = \Sigma \cup \{\rtimes, \ltimes\}$ is the input alphabet,
   - Γ is the output alphabet,
   - $q_0 \in Q$ is the initial state,
   - $F \subseteq Q$ is the set of final states,
   - $\delta : Q \times \Sigma \rightarrow Q \times \Gamma^* \times D$ is the transition function where the direction $D = \{-1, 0, +1\}$.

A *configuration* of a 2-way FST T is an element of $\Sigma_{\ltimes}^* Q \Sigma_{\ltimes}^* \times \Gamma^*$. The meaning of the configuration (wqx, u) is that the input to T is wx and the machine is currently in state q with the read head on the first symbol of x (or has fallen off the right edge of the input tape if $x = \lambda$) and that u is currently written on the output tape.

If the current configuration is $(wqax, u)$ and $\delta(q, a) = (r, v, 0)$ then the next configuration is $(wrax, uv)$, in which case we write $(wqax, u) \rightarrow (wrax, uv)$. If the current configuration is $(wqax, u)$ and $\delta(q, a) = (r, v, +1)$ then the next configuration is $(warx, uv)$. In this case, we write $(wqax, u) \rightarrow (warx, uv)$. If the current configuration is $(waqx, u)$ and $\delta(q, a) = (r, v, -1)$ then the next configuration is $(wrax, uv)$. We write $(waqx, u) \rightarrow (wrax, uv)$.

The transitive closure of $\rightarrow$ is denoted with $\rightarrow^+$. So if $c \rightarrow^+ c'$ then there exists a finite sequence of configurations $c_1, c_2 \ldots c_n$ with $n > 1$ such that $c = c_1 \rightarrow c_2 \rightarrow \ldots \rightarrow c_n = c'$.

Next we define the string-to-string function that a 2-way FST $T = (Q, \Sigma_{\ltimes}, \Gamma, q_0, F, \delta)$ computes. For each string $w \in \Sigma^*$, $f_T(w) = u \in \Gamma^*$ provided there exists $q_f \in F$ such that $(q_0 \rtimes w \ltimes, \lambda) \rightarrow^+ (\rtimes w \ltimes q_f, u)$. Note that since T is deterministic it follows that if $f_T(w)$ is defined then u is unique.

There are situations where a 2-way FST T crashes on some input w and hence $f_T(w)$ is undefined. If the configuration is (qax, u) and $\delta(q, a) = (r, -1, v)$ then the derivation crashes and the transduction $f_T(ax)$ is undefined. Likewise, if the the configuration is (wq, u) and $q \notin F$

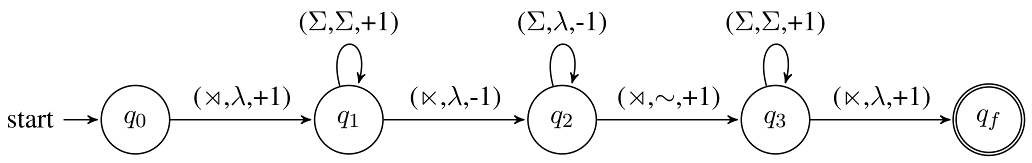

Figure 1: 2-way FST for total reduplication in Indonesian.

Outputting the first copy	Going back to the start of the tape
1. ($\underline{\rtimes}$buku$\ltimes$, λ, q_0 , N/A)	7. ($\rtimes$buk$\underline{\text{u}}\ltimes$, buku, q_2, $q_1 \xrightarrow[-1]{\ltimes:\lambda} q_2$)
2. ($\rtimes\underline{\text{b}}uku\ltimes$, λ , q_1, $q_0 \xrightarrow[+1]{\rtimes:\lambda} q_1$)	8. ($\rtimes$bu$\underline{\text{k}}$u$\ltimes$, buku, q_2, $q_2 \xrightarrow[-1]{\text{u}:\lambda} q_2$)
3. ($\rtimes$b$\underline{\text{u}}$ku$\ltimes$, b, q_1, $q_1 \xrightarrow[+1]{\text{b:b}} q_1$)	9. ($\rtimes$b$\underline{\text{u}}$ku$\ltimes$, buku, q_2, $q_2 \xrightarrow[-1]{\text{k}:\lambda} q_2$)
4. ($\rtimes$bu$\underline{\text{k}}$u$\ltimes$, bu, q_1, $q_1 \xrightarrow[+1]{\text{u:u}} q_1$)	10. ($\rtimes\underline{\text{b}}uku\ltimes$, buku, q_2, $q_2 \xrightarrow[-1]{\text{u}:\lambda} q_2$)
5. ($\rtimes$buk$\underline{\text{u}}\ltimes$, buk, q_1, $q_1 \xrightarrow[+1]{\text{k:k}} q_1$)	11. ($\underline{\rtimes}$buku$\ltimes$, buku, q_2, $q_2 \xrightarrow[-1]{\text{b}:\lambda} q_2$)
6. ($\rtimes$buku$\underline{\ltimes}$, buku, q_1, $q_1 \xrightarrow[+1]{\text{u:u}} q_1$)	
Outputting the second copy	
12. ($\rtimes\underline{\text{b}}uku\ltimes$, buku$\sim$, q_3, $q_2 \xrightarrow[+1]{\rtimes:\sim} q_3$)	15. ($\rtimes$buk$\underline{\text{u}}\ltimes$, buku$\sim$buk, q_3, $q_3 \xrightarrow[+1]{\text{k:k}} q_3$)
13. ($\rtimes$b$\underline{\text{u}}$ku$\ltimes$, buku$\sim$b, q_3, $q_3 \xrightarrow[+1]{\text{b:b}} q_3$)	16. ($\rtimes$buku$\underline{\ltimes}$, buku$\sim$buku, q_3, $q_3 \xrightarrow[+1]{\text{u:u}} q_3$)
14. ($\rtimes$bu$\underline{\text{k}}$u$\ltimes$, buku$\sim$bu, q_3, $q_3 \xrightarrow[+1]{\text{u:u}} q_3$)	17. ($\rtimes$buku$\ltimes$, buku$\sim$buku, q_f, $q_3 \xrightarrow[+1]{\ltimes:\lambda} q_f$)

Table 1: Derivation of /buku/→[buku∼buku].

then the transducer crashes and the transduction f_T is undefined on input w.

There is one more way in which f_T may be undefined for some input. The input may cause the transducer to go into an infinite loop.[5] This occurs for input $wx \in \Sigma^*_{\ltimes}$ whenever there exist $q \in Q$ and $u, v \in \Gamma^*$ such that $(q_0wx, \lambda) \to^+ (wqx, u) \to^+ (wqx, uv)$.

Further information on the computational properties of 2-way FSTs can be found in Filiot and Reynier (2016). Compared to 1-way FSTs, there's relatively little work on complexity metrics for 2-way FSTs, but see Baschenis et al. (2016). See Dolatian and Heinz (2018a,b) for complexity results when using 2-way FSTs for reduplication.

3 The RedTyp Database and Python implementation of 2-way FSTs

This section details the features and organization of the RedTyp database. We likewise discuss our Python implementation of 2-way FSTs which acts as a supplement to the database.

The GitHub page for RedTyp includes the SQL file for the database, a single Python file that can read and implement the 2-way FSTs in the database, and a README textfile for instructions.

Figure 2 shows a simple Entity-Relation diagram (Elmasri and Navathe, 2010) for RedTyp. It consists of two entities: *2-way FST* for representing 2-way FSTs, and *Morpheme* for representing individual reduplicative processes in a language. They engage in a many-to-one *Match* relationship such that every *morpheme* is modeled by one *2-way FST* but a single *2-way FST* can model many reduplicative *morphemes* or reduplicative processes found within and across languages.

For example in Sundanese (Moravcsik, 1978), total reduplication expresses the meaning "not even one X" (ibid., p.301) in addition to intensity (ibid., p.321) as shown in (5)[6] and (6). Thus, in Sundanese there are two distinct morphemes, both of which are modeled with a single 2-way FST for total reduplication.

5.	kali	→ sakali∼kali
	'time'	→ 'not even once'

[5] In RedTyp, all 2-way FSTs were checked to make sure they do not cause infinite loops.

[6] Moravcsik (1978, 301) calls the string 'sa' a prefix. It is not reduplicated.

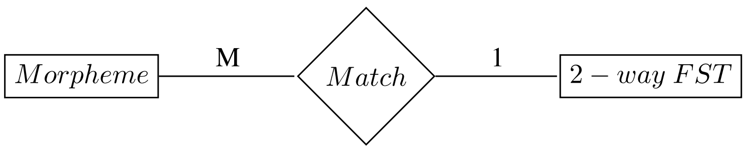

Figure 2: The Entity-Relation diagram for RedTyp.

6. hayaŋ	→	hayaŋ∼hayaŋ
‘want’	→	‘want very much’

We list and describe the attributes of each table in RedTyp below.

The 2-way FST table:

- **ID:** *string or varchar 200*
  A small descriptive name for the type of function modeled by the 2-way FST. This acts as the primary key for a **2-way FST** entry.
- **Example data:** *string or text*
  A table with example inputs and outputs, written in Markdown.
- **Description:** *string or text*
  Description of 2-way FST in terms of prose. This is written in Markdown.
- **FST diagram:** *text*
  LaTeX code for the 2-way FST diagram. It uses the following LaTeX packages/settings.

```
\usepackage{tipa}
\usepackage{tikz}
\usetikzlibrary{arrows,automata,
shapes,positioning}
```

- **FST Recipe:** *text*
  A text file that describes the alphabet and transitions for the FST in a shorthand that is human-readable and that can be read by our Python implementation.
- **FST Code:** *text*
  A text file that lists the transitions for the 2-way FST based on the shorthand description in the FST recipe. This code was created with our Python implementation of 2-way FSTs.
- **Language specific:** *Boolean*
  This field specifies if the 2-way FST entry models a reduplicative process that is unique to a specific morpheme in one language (1) or if it is general enough to model many morphemes in possibly different languages (0).

The Morpheme table:

- **Morpheme ID:** *string or varchar 200*
  The general format for a morpheme ID is ISO-FUNC-FORM where ISO is the ISO code for the morpheme’s language (or the full language name if no ISO exists), FUNC is the semantic function of the morpheme based on the Leipzig glossing system[7] (otherwise the full name if the glossing abbreviation cannot be found), and the general form of the reduplicative process that is part of this morpheme. This is the primary key for the **Morpheme** entry.
- **Language:** *string or varchar 200*
  Name of the language that has this morpheme.
- **Function:** *string or varchar 200*
  The semantic function of the reduplicative morpheme.
- **Default form name:** *string or varchar 200*
  Assuming that the reduplicative morpheme has a default shape, this has the general label for it, e.g. total reduplication, initial CV, etc.
- **Sources** *text*
  Bibliographic sources used.
- **Description**: *text*
  Prosaic description of the data, patterns, and surface shapes of the morpheme, written in Markdown.
- **Data:** *text*
  A table with example inputs and outputs, written in Markdown.
- **Ambiguity:** *text*
  Description of any ambiguities present in the description of this morpheme in the bibliographic references which we used.
- **Shows allomorphy:** *Boolean*
  Specifies if the morpheme has multiple, diverse surface patterns or shapes (1) or if it has one basic shape (0).[8]
- **Has segmentalism:** *Boolean*
  Specifies if the morpheme includes any segmentalism or fixed segments (1), including

[7]The glossing rules can be found on: www.eva.mpg.de/lingua/resources/glossing-rules.php.

[8]It is difficult to give a consistent treatment for how to give a definite value to this attribute in some cases.

both phonological fixed segments (epenthetic vowels) or morphological fixed segments (affixes like *shm-*), or none at all (0).

- **Deeply searched:** *Boolean*
  There are some patterns that we only analyzed with secondary sources such as surveys (not deeply, thus 0); while others we analyzed using primary sources such as in-depth theoretical treatments or descriptive grammars (deeply analyzed, thus 1).
- **Possible vowel modifications:** *string or varchar 200*
  Some reduplicative process have the reduplicant vowel undergo various modifications: lengthening, shortening, diphthong reduction. This lists any vowel changes seen.
- **Morpho-phonological subconstituent involved:** *string or varchar 200*
  Some reduplicative process may target specific morphological/prosodic subconstituents, usually roots. This lists any such subconstituents involved, if any.
- **Involves affixes?:** *Boolean*
  Some reduplicative processes are either caused by or accompanied by affixes (1), while some do not involve any affixes (0).
- **Affix incorporation?:** *Boolean*
  Sometimes an affix not only triggers reduplication but it also undergoes it with the stem it attached to (1); otherwise (0).
- **Involves phonological processes?:** *Boolean*
  Sometimes a reduplicative process will involve a phonological process or opacity (1); some cases do not involve any interactions with phonology (0).
- **Data too ambiguous?:** *Boolean*
  Sometimes the source was too vague to specify the reduplicative process so no 2-way FST is provided (1). Fortunately, most cases were sufficiently described (0).
- **Miscellaneous:** *string or varchar 200*
  Miscellaneous information about this reduplicative process.

The Match table:

- **2-way FST ID:** *string or varchar 200*
  Foreign key that references the 2-way FST ID.
- **Morpheme ID:** *string or varchar 200*
  Foreign key that references the morpheme ID.

In addition to the database itself, we provide a Python script implementing 2-way FSTs. The Python program takes as input an ***FST recipe*** (that can be retrieved from the database) and a user-made text file ***input_strings*** that contains a list of strings for the 2-way FST to process. It creates two textfiles: ***output_transitions*** which contains a list of transition arcs that make up the 2-way FST, and ***output_strings*** which provides the output of every entry in the ***input strings*** using the 2-way FST.

The Python implementation can likewise take as input two textfiles: one is a list of states and transition arcs not written in shorthand (***input_transitions***), and the other is a list of input strings (***input_strings***). It will run the 2-way FST described in ***input_transitions*** and create a textfile ***output_strings*** that provides the output of each input in ***input_strings***.

Because the 2-way FSTs generally had a small number of states, we validated their correctness through manual inspection and by testing them with key examples. Our online repository contains detailed instructions and examples on how to design 2-way FSTs, extract 2-way FSTs from the database, and run them with user-made input.

Lastly, it should be noted that many of the 2-way FSTs in RedTyp define partial functions. As such, they are undefined for some logically possible inputs. For example, a 2-way FST for an initial-C reduplication is only defined for C-initial inputs. As such, it returns an error when the input is V-initial such as 'apa'.

4 Discussion

4.1 Comparison with other computational tools

Within computational morphology, many formal devices have been proposed to handle reduplication. Some of the computational models motivated by reduplication include Multiple Context-Free Grammars (MCFGs) (Seki et al., 1991, 1993; Albro, 2005) and pushdown acceptors augmented with queues (Savitch, 1989). Our reasons for using 2-way FSTs instead of MCFGs and pushdown acceptors are as follows.[9]

[9]The only other existing alternative to 2-way FSTs to our knowledge are pushdown transducers, which augment 1-way transducers with a stack (Allauzen and Riley, 2012). A full comparison between pushdown transducers and 2-way FSTs for modeling processes like reduplication in natural languages is an important exercise for future research.

First, unlike 2-way FSTs, MCFGs and push-down automata do not transform an input string to an output string; rather, they recognize sets of strings that may include a copying component such as the formal language $\{ww \mid w \in \Sigma^*\}$. Although Albro (2005) extended finite-state OT with MCFGs to model reduplicative processes in Malagasy, Albro's method does not provide a general machine model. We thought it judicious to provide, as resources, well-studied machine models in computer science as opposed to extending an existing formalism ourselves.

This brings us to our second reason: the class of transductions definable with 2-way FSTs are exactly the class of transductions definable with Monadic Second Order (MSO) Logic (Engelfriet and Hoogeboom, 2001). Logic, automata, and the relationship between them are foundational subjects in formal language theory (Büchi, 1960; McNaughton and Papert, 1971; Thomas, 1997), though the importance of studying string-to-string translations against this backdrop has only recently been realized (Filiot and Reynier, 2016). Our third reason is practical. As mentioned, 2-way FSTs are easy to write, debug, and implement.

4.2 Comparison with the Graz Database on Reduplication

RedTyp is not the first database for reduplication. That distinction belongs to the Graz Database on Reduplication (GDR), a SQL-database which documents and describes reduplicative processes in 82 languages (Hurch, 2005 ff.).[10]

It is the only other existing database for reduplication to our knowledge. The GDR's language sample is based on the WALS 100-language sample (Dryer and Haspelmath, 2013), but includes other languages "when these contain interesting reduplication types or when the researchers have a particular interest in any language or when there happens to be a considerable amount of information available to the researchers on the reduplication system of any language."

There are three primary distinctions between RedTyp and GDR. First, the information in the databases were collected in different ways. GDR is based on the WALS 100 language sample, and RedTyp is based on linguistic surveys. In terms of coverage and overlap, there are 69 languages in RedTyp that are not in GDR; 60 languages in GDR that are not in RedTyp; and 22 languages common to both. They thus differ in coverage considerably. We speculate this difference is due to the goals and sources of the two databases. The GDR aims to collect examples of reduplicative processes from major and minor world languages in WALS and organize them according to form and function. There is not a specific drive to collect theoretically diverse examples of reduplication. On the other hand, the linguistic surveys and case studies used to develop RedTyp aim to document both common reduplicative processes and theoretically interesting and uncommon reduplicative processes from a morphological and phonological perspective.

Second, users can query GDR online according to pre-made and custom-made linguistically-informed searches, whereas RedTyp has no current presence online. Third, GDR does not contain any computational models of reduplication, whereas RedTyp does. Thus, RedTyp and GDR have different complementary utilities and functionalities. It is likewise an open question if RedTyp misses certain theoretically interesting patterns which are found in GDR.

4.3 Utility of RedTyp

RedTyp provides a set of 2-way FSTs representing copying processes that occur in natural language. This potentially has many uses.

For computational morphologists and phonologists working on computationally modeling or computationally implementing productive reduplicative processes in any language, RedTyp's 2-way FST examples can help them design 2-way FSTs for their own needs. To this end, RedTyp includes instructions to users on how to adapt existing 2-way FSTs by using "recipes".

For formal language theorists and theoretical linguists working on copying, the examples in RedTyp can be studied to identify additional properties, which can lead to new hypotheses about the nature of reduplicative morphology in the world's languages (cf. Heinz (2009)). For researchers interested in learning natural language patterns, RedTyp provides a computationally explicit set of learning targets. In fact, We have used RedTyp ourselves to get typological generalizations and learnability results (Dolatian and Heinz, 2018a).

[10] The reason why we created our own database instead of using GDR was because the GDR was offline during the time we spent collecting data for RedTyp. GDR came back online in 2017, after a hiatus of at least two years. At that time, most of the data collection for RedTyp had been completed.

Finally, our use of 2-way FSTs to model copying in natural languages opens doors on how copying, mimicry, or doubling in other domains may be modeled, including animal communication, bioinformatics and robotics.

4.4 Future Research

RedTyp also provides a focal point for different streams of future research in four general directions: improving RedTyp itself, translating linguistic theories of reduplication into 2-way FSTs, identifying subclasses of 2-way FSTs especially relevant for reduplication, and developing theory and tools to deploy 2-way FSTs in Natural Language Processing (NLP) tasks. We discuss research in both of these areas in turn.

First, although the reduplicative processes in RedTyp are fairly representative, of course more reduplicative morphemes from additional languages ought to be incorporated. We plan to grow RedTyp by exhaustively including more data from surveys such as McCarthy and Prince (1995), Inkelas and Zoll (2005), and Hurch (2005). Second, we plan to consult primary sources to verify the accuracy of the linguistic information in RedTyp. Additionally, we would like to integrate RedTyp with the Graz database (Hurch, 2005 ff.) in some fashion so that researchers interested in reduplication can take full advantage of what both databases offer, both in utility and coverage. Furthermore, an online interface should be developed.

Second, 2-way FSTs appear well-suited to capture the intuition that reduplication consists of actively copying segments from the input into multiple positions of the output. Various theoretical principles on reduplication such templatic association (Marantz, 1982), sensitivity to prominent positions in the input (Raimy, 2000), and multiple access to the same input (Steriade, 1988; Inkelas and Zoll, 2005) can be fruitfully encoded in 2-way FSTs. For results on how 2-way FSTs capture the theory behind reduplication, see Dolatian and Heinz (2018b, In press.).

Third, RedTyp contains 2-way FSTs because they were expressive enough to model both total and partial reduplication exactly and succinctly. However, reduplication does not need the full power of 2-way FSTs. Just as aspects of segmental phonology appear to be circumscribed by subclasses of 1-way FSAs/FSTs (Heinz, 2009, 2010; Chandlee, 2014; Chandlee et al., 2014, 2015; Jardine, 2016; Luo, 2017; Payne, 2017), it is likely that reduplication has stronger computational properties. We are currently investigating some promising subclasses of 2-way FSTs. One subclass in particular is the C-OSL subclass which covers the bulk of reduplicative typology and for which there are learnability results (Dolatian and Heinz, 2018a).

Finally, with respect to developing theory and tools for deploying 2-way FSTs for NLP tasks, we are optimistic. We expect that 2-way FSTs can be incorporated into existing finite-state platforms. First of all, they are straightforward extensions of 1-way FSTs, which facilitates integration. Secondly, they also have a clear logical interpretation (Engelfriet and Hoogeboom, 2001), which bolsters their status as a mathematically and computationally natural formalism. Third is the fact that computationally equivalent formalisms such as Streaming String Transducers (Alur, 2010) have industrial utility and applications (Alur and Černý, 2011; Alur et al., 2014).

Concretely, the next step for this should focus on recognition. The Python program we provided for 2-way FSTs includes code for generation, but not recognition. One route towards recognition may be non-deterministic FSTs (Alur and Deshmukh, 2011; Filiot and Reynier, 2016). Another is to see to what extent pushdown transducers (Allauzen and Riley, 2012) simulate deterministic and non-deterministic 2-way FSTs and vice versa.

To sum up, once the properties of 2-way FSTs are more more fully understood it should be possible to expand existing finite-state processing platforms for morphology, such as xfst (Beesley and Karttunen, 2003), openFST, (Allauzen et al., 2007), foma (Hulden, 2009b), and Pynini (Gorman, 2016), to include them.

5 Conclusion

RedTyp provides a publicly available record of 138 reduplicative processes and morphemes, each of which is implemented precisely with a deterministic 2-way FST. It is striking that so many reduplicative processes can be described compactly with this understudied type of finite-state machine. We hope RedTyp spurs additional research activity into computational and theoretical treatments of reduplication as well as theoretical and applied research on 2-way FSTs as models of morphophonological copying processes.

Acknowledgments

This research is supported by NIH grant #R01-HD087133 to JH. We thank audiences at CLS53, NAPhCX, ICGI, SIGMORPHON, the University of Delaware, and Stony Brook University.

References

Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. 1969. A general theory of translation. *Mathematical Systems Theory*, 3(3):193–221.

Daniel M. Albro. 2005. *Studies in Computational Optimality Theory, with Special Reference to the Phonological System of Malagasy*. Ph.D. thesis, University of California, Los Angeles, Los Angeles.

Cyril Allauzen and Michael Riley. 2012. A pushdown transducer extension for the OpenFst library. In *Implementation and Application of Automata*, pages 66–77, Berlin, Heidelberg. Springer.

Cyril Allauzen, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. 2007. OpenFst: A general and efficient weighted finite-state transducer library. In *Implementation and Application of Automata*, pages 11–23, Berlin, Heidelberg. Springer.

Rajeev Alur. 2010. Expressiveness of streaming string transducers. In *Proceedings of the 30th Annual Conference on Foundations of Software Technology and Theoretical Computer Science,*, volume 8, page 112.

Rajeev Alur and Jyotirmoy V. Deshmukh. 2011. Nondeterministic streaming string transducers. In *Automata, Languages and Programming*, pages 1–20, Berlin, Heidelberg. Springer.

Rajeev Alur, Adam Freilich, and Mukund Raghothaman. 2014. Regular combinators for string transformations. In *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, CSL-LICS '14, pages 9:1–9:10, New York, NY, USA. ACM.

Rajeev Alur and Pavol Černý. 2011. Streaming transducers for algorithmic verification of single-pass list-processing programs. In *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '11, pages 599–610, New York, NY, USA. ACM.

Félix Baschenis, Olivier Gauwin, Anca Muscholl, and Gabriele Puppis. 2016. Minimizing Resources of Sweeping and Streaming String Transducers. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*, volume 55 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 114:1–114:14, Dagstuhl, Germany. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

Kenneth R Beesley and Lauri Karttunen. 2003. *Finite-state morphology: Xerox tools and techniques*. CSLI Publications.

J. Richard Büchi. 1960. Weak second-order arithmetic and finite automata. *Mathematical Logic Quarterly*, 6(1-6):66–92.

Jane Chandlee. 2014. *Strictly Local Phonological Processes*. Ph.D. thesis, University of Delaware, Newark, DE.

Jane Chandlee. 2017. Computational locality in morphological maps. *Morphology*, pages 1–43.

Jane Chandlee, Rémi Eyraud, and Jeffrey Heinz. 2014. Learning strictly local subsequential functions. *Transactions of the Association for Computational Linguistics*, 2:491–503.

Jane Chandlee, Rémi Eyraud, and Jeffrey Heinz. 2015. Output strictly local functions. In *Proceedings of the 14th Meeting on the Mathematics of Language (MoL 2015)*, pages 112–125, Chicago, USA.

Jane Chandlee and Jeffrey Heinz. 2012. Bounded copying is subsequential: Implications for metathesis and reduplication. In *Proceedings of the 12th Meeting of the ACL Special Interest Group on Computational Morphology and Phonology*, SIGMORPHON '12, pages 42–51, Montreal, Canada. Association for Computational Linguistics.

Yael Cohen-Sygal and Shuly Wintner. 2006. Finite-state registered automata for non-concatenative morphology. *Computational Linguistics*, 32(1):49–82.

Abigail C Cohn. 1989. Stress in Indonesian and bracketing paradoxes. *Natural language & linguistic theory*, 7(2):167–216.

Hossep Dolatian and Jeffrey Heinz. 2018a. Learning reduplication with 2-way finite-state transducers. In *Proceedings of Machine Learning Research: International Conference on Grammatical Inference*, volume 93 of *Proceedings of Machine Learning Research*, pages 67–80, Wroclaw, Poland.

Hossep Dolatian and Jeffrey Heinz. 2018b. Modeling reduplication with 2-way finite-state transducers. In *Proceedings of the 15th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*, Brussells, Belgium. Association for Computational Linguistics.

Hossep Dolatian and Jeffrey Heinz. In press. Reduplication with finite-state technology. In *Proceedings of the 53rd Annual Meeting of the Chicago Linguistics Society*.

Laura J Downing. 2000. Morphological and prosodic constraints on Kinande verbal reduplication. *Phonology*, 17(01):1–38.

Matthew S. Dryer and Martin Haspelmath, editors. 2013. *WALS Online*. Max Planck Institute for Evolutionary Anthropology, Leipzig.

Ramez Elmasri and Shamkant Navathe. 2010. *Fundamentals of Database Systems*, 6th edition. Addison-Wesley Publishing Company, USA.

Joost Engelfriet and Hendrik Jan Hoogeboom. 2001. MSO definable string transductions and two-way finite-state transducers. *ACM Trans. Comput. Logic*, 2(2):216–254.

Martin Everaert, Simon Musgrave, and Alexis Dimitriadis. 2009. *The use of databases in cross-linguistic studies*, volume 41. Walter de Gruyter.

Emmanuel Filiot and Pierre-Alain Reynier. 2016. Transducers, logic and algebra for functions of finite words. *ACM SIGLOG News*, 3(3):4–19.

R.W.N. Goedemans, H.G. van der Hulst, and E.A.M. Visch. 1996. *Stress Patterns of the World Part 1: Background*. HIL Publications II. Holland Academic Graphics. The Hague.

Kyle Gorman. 2016. Pynini: A python library for weighted finite-state grammar compilation. In *Proceedings of the SIGFSM Workshop on Statistical NLP and Weighted Automata*, pages 75–80. Association for Computational Linguistics.

Jeffrey Heinz. 2009. On the role of locality in learning stress patterns. *Phonology*, 26(2):303–351.

Jeffrey Heinz. 2010. Learning long-distance phonotactics. *Linguistic Inquiry*, 41(4):623–661.

Jeffrey Heinz, Rob Goedemans, and Harry van der Hulst, editors. 2016. *Dimensions of Phonological Stress*. Cambridge University Press.

Mans Hulden. 2009a. *Finite-state machine construction methods and algorithms for phonology and morphology*. Ph.D. thesis, The University of Arizona, Tucson, AZ.

Mans Hulden. 2009b. Foma: a finite-state compiler and library. In *Proceedings of the Demonstrations Session at EACL 2009*, pages 29–32. Association for Computational Linguistics.

Mans Hulden and Shannon T Bischoff. 2009. A simple formalism for capturing reduplication in finite-state morphology. In *Proceedings of the 2009 conference on Finite-State Methods and Natural Language Processing: Post-proceedings of the 7th International Workshop FSMNLP 2008*, pages 207–214, Amsterdam. IOS Press.

Bernhard Hurch, editor. 2005. *Studies on reduplication*. 28. Walter de Gruyter, Berlin.

Bernhard Hurch. 2005 ff. Graz database on reduplication. Last accessed 10-26-2017 from `http://reduplication.uni-graz.at/redup/`.

Sharon Inkelas and Laura J Downing. 2015. What is reduplication? Typology and analysis part 1/2: The typology of reduplication. *Language and Linguistics Compass*, 9(12):502–515.

Sharon Inkelas and Cheryl Zoll. 2005. *Reduplication: Doubling in Morphology*. Cambridge University Press, Cambridge.

Adam Jardine. 2016. Computationally, tone is different. *Phonology*, 33(2):247–283.

Huan Luo. 2017. Long-distance consonant agreement and subsequentiality. *Glossa: a journal of general linguistics*, 2(1):125.

Alec Marantz. 1982. Re reduplication. *Linguistic inquiry*, 13(3):435–482.

John J McCarthy and Alan Prince. 1995. Faithfulness and reduplicative identity. In Jill N. Beckman, Laura Walsh Dickey, and Suzanne Urbanczyk, editors, *Papers in Optimality Theory*. Graduate Linguistic Student Association, University of Massachusetts, Amherst, MA.

Robert McNaughton and Seymour A Papert. 1971. *Counter-Free Automata (MIT research monograph no. 65)*. The MIT Press.

Mehryar Mohri. 1997. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311.

Edith Moravcsik. 1978. Reduplicative constructions. In Joseph Greenberg, editor, *Universals of Human Language*, volume 1, pages 297–334. Stanford University Press, Stanford, California.

Amanda Payne. 2017. All dissimilation is computationally subsequential. *Language: Phonological Analysis*, 93(4):e353–e371.

Eric Raimy. 2000. *The Phonology and Morphology of Reduplication*. Berlin: Mouton de Gruyter.

Eric Raimy. 2011. Reduplication. In Marc van Oostendorp, Colin Ewen, Elizabeth Hume, and Keren Rice, editors, *The Blackwell companion to phonology*, volume 4, pages 2383–2413. Wiley-Blackwell, Malden, MA.

Brian Roark and Richard Sproat. 2007. *Computational Approaches to Morphology and Syntax*. Oxford University Press, Oxford.

Carl Rubino. 2005. Reduplication: Form, function and distribution. In *Studies on reduplication*, pages 11–29. Mouton de Gruyter, Berlin.

Carl Rubino. 2013. *Reduplication*. Max Planck Institute for Evolutionary Anthropology, Leipzig.

Walter J Savitch. 1989. A formal model for context-free languages augmented with reduplication. *Computational Linguistics*, 15(4):250–261.

Hiroyuki Seki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. 1991. On multiple context-free grammars. *Theoretical Computer Science*, 88(2):191–229.

Hiroyuki Seki, Ryuichi Nakanishi, Yuichi Kaji, Sachiko Ando, and Tadao Kasami. 1993. Parallel multiple context-free grammars, finite-state translation systems, and polynomial-time recognizable subclasses of lexical-functional grammars. In *Proceedings of the 31st annual meeting on Association for Computational Linguistics*, pages 130–139. Association for Computational Linguistics.

Jeffrey Shallit. 2008. *A Second Course in Formal Languages and Automata Theory*, 1 edition. Cambridge University Press, New York, NY, USA.

Donca Steriade. 1988. Reduplication and syllable transfer in Sanskrit and elsewhere. *Phonology*, 5(1):73–155.

Wolfgang Thomas. 1997. Languages, automata, and logic. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 389–455. Springer-Verlag New York, Inc., New York, NY, USA.

Suzanne Urbanczyk. 2007. Themes in phonology. *The Cambridge Handbook of Phonology, edited by Paul de Lacy*, pages 473–493.